

**APPLICATION FOR A UNITED STATES PATENT**  
**UNITED STATES PATENT AND TRADEMARK OFFICE**

(CASE No. 99,447)

Title: **METHOD AND APPLICATION PROGRAMMING INTERFACE FOR  
ASSIGNING MULTIPLE NETWORK ADDRESSES**

5

10 Inventors: Michael S. Borella, a citizen of the United States of America, and a resident of  
Naperville, Illinois; and

Nurettin B. Beser, a citizen of Turkey, and a resident of Evanston, Illinois.

15  
20 Assignee: 3Com Corporation  
5400 Bayfront Plaza  
Santa Clara, CA 95052

## FIELD OF INVENTION

The present invention relates to communications in data networks. More specifically, it relates to a method and an application programming interface for assigning multiple network addresses.

## BACKGROUND OF THE INVENTION

The Internet Protocol ("IP") and Transmission Control Protocol ("TCP") are rapidly becoming the lingua franca of modern data networks. Currently, host protocol stacks allow a single host to support multiple physical and logical data-link interfaces.

These interfaces may be either software or a combination of hardware and software. A data-link interface is typically a device that permits a host to communicate with another entity. Data-link interfaces may be active for as long as the host is running, or they may activate and deactivate dynamically. An example of a data-link interface is an Ethernet interface which consists of the Ethernet card along with a device driver. Typically the Ethernet interface becomes active upon system boot and remains active until the host computer is shut off.

Data-link interfaces typically represent an IP address that is bound to a data-link device. Communication via these interfaces occurs through device-independent calls to a socket application programming interface ("API"). As is known in the art, a socket is an endpoint, in a host computer's protocol stack, for communicating over a data network. The socket API provides the capability for application programs and their processes to access communications protocols automatically. The socket API exists logically above a transport layer for the TCP/IP

stack, but below an application layer. The socket API, which is well known to those skilled in the art, is discussed in UNIX on-line manuals as well as many textbooks.

At present, all processes typically bind to one common IP address. When a process initiates an IP communication, the protocol stack binds the common IP address to the process. In this sense, all processes typically communicate over the same IP interface. Data traffic to or from one application is typically distinguished from data traffic to or from another application by a transport-layer parameter such as a TCP or User Datagram Protocol ("UDP") port. For some processes, however, having one common IP address may be restrictive.

Currently, protocol stacks may support multiple IP interfaces in a limited sense. A host with more than one physical interface may require more than one IP address. For example, a host computer may communicate using IP packets over a Point-to-point Protocol ("PPP") connection via a modem while simultaneously communicating using IP packets over an Ethernet connection. In this case, the Ethernet interface and the PPP interface would be associated with separate IP addresses. The processes on the host computer, however, have these separate IP addresses in common. Any process that communicates over the modem does so using the common IP address for the PPP connection. Similarly, any process that communicates over the Ethernet connection does so using the common IP address for the Ethernet connection.

Future communication systems, however, may require that a protocol stack is capable of supporting multiple IP addresses in a broader sense: different processes on the same host are associated with different IP addresses on the same physical interface. Processes may request a new IP interface with a new IP address rather than share a single IP address with all other

processes. Instead of distinguishing traffic to and from processes by port numbers, the traffic may be distinguished by IP address.

It is therefore desirable to provide a method for binding multiple IP addresses to the same process. Multiple processes on the same host may then be assigned different IP addresses that

5 are distinguishable at an application layer.

0042205022400  
09511735

## SUMMARY OF THE INVENTION

In accordance with preferred embodiments of the present invention, methods for assigning multiple network addresses are provided. One aspect of the invention includes a method for using multiple network addresses for interprocess communication through a common physical layer. The method includes creating a first interprocess communication data structure associated with a first network address on a first network device. A first communication is established between the first network device and a second network device using the first interprocess communication data structure and the first network address. The first communication passes through the common physical layer for the first network device. A second interprocess communication data structure associated with a second network address is created on the first network device. The second network address is different from the first network address. A second communication is created between the first network device and a third network device using the second interprocess communication data structure and the second network address. The second communication also passes through the common physical layer for the first network device.

The method may help overcome limitations in having one network address common to every process on the host computer. With this method, each process may create its own interprocess communication data structure for communicating over a data network. In turn, each interprocess communication data structure may be associated with its own network address.

The foregoing and other features and advantages of preferred embodiments of the present invention will be more readily apparent from the following detailed description, which proceeds with references to the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention are described with reference to the following drawings, wherein:

FIG. 1 is a block diagram illustrating a network system;

5 FIG. 2 is a block diagram illustrating a protocol stack for a network device;

FIG. 3 is a block diagram illustrating a typical stack implementation;

FIG. 4 is a block diagram illustrating a stack implementation with multiple network addresses; and

FIG. 5 is a flow diagram illustrating a method for using multiple network addresses.

004220-02400  
0951735-02400

## DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. 1 is a block diagram illustrating an exemplary data network 10 for an illustrative embodiment of the present invention. The data network 10 includes a backbone network 12 (e.g. the Internet), a first network device 14, and a second network device 16. The backbone network 12 may be public in the sense that it may be accessible by many users who may monitor communications on it. Additionally, although only one is illustrated, there may be multiple local area networks ("LANs") 20 coupled to the backbone network 12. Data packets may be transferred to/from the first network device 14 and the second network device 16 over the backbone network 12. For example, the devices may be assigned public network addresses on the Internet. A data channel between the first network device 14 and the second network device 16 may include routers or gateways (24, 26). However, other data network types and network devices can also be used and the present invention is not limited to the data network and network devices described for an illustrative embodiment.

For example, the routers (24, 26) may be edge routers. An edge router routes data packets between one or more networks such as a public network (e.g. backbone network 12) and a private network (e.g. LAN 20). Edge routers are commercially available from numerous sources, including those provided by 3Com Corporation of Santa Clara, California, Cisco Systems of San Jose, California, Lucent Technologies of Murray Hill, New Jersey, Lucent subsidiaries including Livingston Enterprises, Inc. of Pleasanton, California, and Ascend Communications of Alameda, California, and others.

In one exemplary preferred embodiment of the present invention, the first 14 and second 16 network devices are telephony devices or bulk data devices. Bulk data devices include Web-

be associated with the QoS of the application. The IP 58 address and QoS of the application may be conveyed to the edge router (24,26) to establish a layer 3 switching channel for the application. Therefore, each communication is over a virtual channel between applications associated with a particular QoS and a unique IP 58 address.

5 It should be understood that the programs, processes, methods, systems and apparatus described herein are not related or limited to any particular type of computer apparatus (hardware or software), unless indicated otherwise. Various types of general purpose or specialized computer apparatus may be used with or perform operations in accordance with the teachings described herein.

10 In view of the wide variety of embodiments to which the principles of the invention can be applied, it should be understood that the illustrated embodiments are exemplary only, and should not be taken as limiting the scope of the present invention. For example, the steps of the flow diagrams may be taken in sequences other than those described, and more or fewer elements or components may be used in the block diagrams.

15 The claims should not be read as limited to the described order or elements unless stated to that effect. In addition, use of the term "means" in any claim is intended to invoke 35 U.S.C. §112, paragraph 6, and any claim without the word "means" is not so intended. Therefore, all implementations that come within the scope and spirit of the following claims and equivalents thereto are claimed as the invention.



TV sets and decoders, interactive video-game players, or personal computers running multimedia applications. Telephony devices include Voice over Internet Protocol ("VoIP") devices (portable or stationary) or personal computers running facsimile or audio applications. However, the ends of the data flow may be other types of network devices and the present invention is not restricted to telephony or bulk data devices.

Network devices and routers for preferred embodiments of the present invention include network devices that can interact with network system 10 based on standards proposed by the Institute of Electrical and Electronic Engineers ("IEEE"), International Telecommunications Union-Telecommunication Standardization Sector ("ITU"), Internet Engineering Task Force ("IETF"), or Wireless Application Protocol ("WAP") Forum. However, network devices based on other standards may also be used. IEEE standards can be found on the World Wide Web at the Universal Resource Locator ("URL") "www.ieee.org." The ITU, (formerly known as the CCITT) standards can be found at the URL "www.itu.ch." IETF standards can be found at the URL "www.ietf.org." The WAP standards can be found at the URL "www.wapforum.org." Such standards, and the organizations that establish them, are well known to those skilled in the art.

It will be appreciated that the configuration and devices of FIG. 1 are for illustrative purposes only and the present invention is not restricted to network devices such as edge routers, and telephony or bulk data devices. As will be recognized by those skilled in the art, many other network devices are possible. Moreover, the configuration of data network 10 is not restricted to one backbone network 12 and one LAN 20 as shown in FIG. 1. Many different configurations of

the data network 10 with multiple data networks and/or multiple local area networks at various positions in the data network configuration 10 are possible.

An operating environment for network devices of the present invention includes a processing system with at least one high speed Central Processing Unit ("CPU") and a memory.

5 In accordance with the practices of persons skilled in the art of computer programming, the preferred embodiments are described below with reference to acts and symbolic representations of operations or instructions that are performed by the processing system, unless indicated otherwise. Such acts and operations or instructions are referred to as being "computer-executed," "CPU executed," or "executable."

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652

readable medium, which exist exclusively on the processing system or be distributed among multiple interconnected processing systems that may be local or remote to the processing system.

### Network device protocol stack

FIG. 2 is a block diagram illustrating a protocol stack 50 for network devices, such as devices 14 and 16 in FIG. 1, in data network 10. The protocol stack 50 is typically executable code and data structures associated with a kernel for an operating system of the network device (14,16). The code resides in memory locations associated with the kernel and directs a CPU or CPUs to format and exchange network communications properly. The data structures are portions of memory that are used by the protocol stack code to retain dynamic and static values, and are also accessible and controllable by the kernel.

As is known in the art, the Open System Interconnection ("OSI") model may describe computer networks. The OSI model consists of seven layers including from lowest-to-highest, a physical, data-link, network, transport, session, presentation, and application layer. The physical layer exchanges bits over a communication link. The data link layer exchanges error free frames of data. The network layer exchanges and routes data packets.

The lowest layer of the protocol stack is the physical layer. The physical layer includes the physical media interfaces 52 that place signals on transmission media such as wires, coaxial cable, optical fiber, or transmit them as electromagnetic waves. The physical media interfaces 52 also read signals from the transmission media and present them to the data-link layer.

In the data-link layer is a Medium Access Control ("MAC") layer 54. As is known in the art, the MAC layer 54 controls access to a transmission medium via the physical layer. For more information on the MAC layer protocol 54 see IEEE 802.3 for Ethernet and IEEE 802.14 for

cable modems. However, other MAC layer protocols 54 could also be used and the present invention is not limited to IEEE 802.3 or IEEE 802.14.

Above the data-link layer is an Internet Protocol ("IP") layer 58. The IP layer 58, hereinafter IP 58, roughly corresponds to OSI layer 3, the network layer, but is typically not defined as part of the OSI model. As is known in the art, the IP 58 is a message addressing and delivery protocol designed to route traffic within a network or between networks. For more information on the IP 58 see IETF RFC-791, the contents of which are incorporated herein by reference.

The Internet Control Message Protocol ("ICMP") layer 56 is used for network management. The main functions of the ICMP layer 56, hereinafter ICMP 56, include error reporting, reachability testing (e.g., "pinging") congestion control, route-change notification, performance, subnet addressing and others. Since the IP 58 is an unacknowledged protocol, datagrams may be discarded and the ICMP 56 is used for error reporting. For more information on the ICMP 56 see IETF RFC-792, the contents of which are incorporated herein by reference.

Above the IP 58 and the ICMP 56 is a transport layer with a User Datagram Protocol layer 60 ("UDP"). The UDP layer 60, hereinafter UDP 60, roughly corresponds to OSI layer 4, the transport layer, but is typically not defined as part of the OSI model. As is known in the art, the UDP 60 provides a connectionless mode of communications with datagrams. For more information on the UDP 60 see IETF RFC-768, the contents of which are incorporated herein by reference. The transport layer may also include a connection-oriented Transmission Control Protocol ("TCP") layer 62. For more information on TCP see IETF RFC-793 and IETF RFC-1323, the contents of which are incorporated herein by reference.

Above the transport layer is an application layer where the application programs that carry out desired functionality for a network device reside. For example, the application programs for the network device 16 may include a printer application program, while application programs for the network device 14 may include a facsimile application program.

5 In the application layer are typically a Dynamic Host Configuration Protocol ("DHCP") layer 66 and a File Transfer Protocol ("FTP") layer 68. The DHCP layer 66 is a protocol for passing configuration information to and from hosts on an IP 58 network. For more information on the DHCP layer 66 see IETF RFC-2131, the contents of which are incorporated herein by reference. The FTP layer 68 is a file transfer protocol used to download files and configuration information. For more information on the FTP layer 68 see IETF RFC-959, the contents of which are incorporated herein by reference. Processes also reside in the application layer. While FIG. 2 identifies a five layer OSI model, those skilled in the art will recognize that more or fewer protocol layers may be used in the protocol stack 50.

### Network address interfaces

FIG. 3 is a block diagram illustrating a typical stack implementation 80. The diagram depicts a typical TCP/IP or UDP/IP stack implementation in many operating systems. One or more processes 82-86 may each connect to one or more sockets 88-92. As is known to those skilled in the art, an application is a program that performs a useful task. An application may include many processes, each of which is a set of instructions for directing a CPU to perform a specific task.

20 Additionally, each process 82-86 may connect to one or more sockets 88-92 and each socket 88-92 may connect to one or more processes 82-86. In general, a process may connect to

more than one socket. Also, the processes within an application may simultaneously be associated with the same socket. For example: Process A 82 is connected to Socket #2 90; Process B 84 is connected to both Socket #1 88 and Socket #2 90; and Process C 86 is connected to Socket #2 90 and Socket #3 92.

5 Each socket includes a source IP 58 address as well as a source TCP 62 or UDP 60 port. Typically, each of the source IP 58 addresses in the sockets 88-92 are actually pointers to a host's single IP 58 address for its network address interface 94, which in turn is bound to a data-link interface 96. The IP 58 address typically resides in memory associated with a kernel of an operating system for the host network device. All Sockets 88-92 share the same IP 58 address and data-link layer, and communicate over the same physical medium.

As will be further described below, a function call may specify the IP 58 address for the socket, provided that this IP 58 address is a valid IP 58 address for the network address interface 94. More generally, however, the function call does not specify the IP 58 address. Instead, it allows the kernel of the operating system for the host network device to choose the (known) IP 58 address for the network address interface 94. Also, by calling the socket creation and binding functions, the process may specify a TCP 62 or UDP 60 port number (especially if the application uses a well-known port number as is familiar to those skilled in the art). Alternatively, the process may leave the choice of port up to the kernel of the operating system (the latter being referred to as an "ephemeral" port in the art).

20 An API for the sockets 88-92 includes reentrant functions for creating sockets and binding them to interfaces. As is known to those skilled in the art, a function is a self-contained set of instructions for carrying out a particular task. When a process calls a reentrant function,

the kernel of the operating system for the network device directs the CPU to execute the code of the function. After executing the code of the function, the kernel instructs the CPU to reenter the calling process, to return a result at the point where the calling process called the function, and to continue executing the code of the calling process. For more information on socket functions, see "The Design of the Unix Operating System", by Maurice J. Bach, Prentice-Hall, 1990, the contents of which are incorporated herein by reference. Generally, processes running on the host network device call the reentrant functions for the socket API.

In operation, a first process on a first network device 14, e.g. Process B 84 of FIG. 3, exchanges data with a second process (not shown) on a second network device 16 through an interprocess communication. An example of an interprocess communication is a socket-to-socket virtual data link. Data from the first process transfers to an appropriate interprocess communication data structure, e.g. Socket #1 88. The interprocess communication data structure is a portion of memory in the first network device 14 that is associated with the protocol stack 50 for the first network device 14. When the first process places data in this data structure, the kernel of the operating system for the first network device 14 encapsulates the data and transmits it as a packet to the second network device 16. Once the packet reaches the second network device 16, the kernel of the second network device 16 extracts the data and places it in another interprocess communication data structure. The other interprocess communication data structure is associated with the second process, e.g. a socket at the other end of the interprocess communication. The second process may access the other data structure to receive the data from the first process. The operating system creates and configures interprocess communications data structures through calls to an API.

## API for a single network address

A call to a reentrant function called "socket" creates a socket by setting up a data structure in the host computer's memory. A "socket" call may take the following form:

$$sd = \text{socket}(\text{format}, \text{type}, \text{protocol}) \quad (1)$$

- 5 The socket descriptor, "sd", is typically a handle that is used by the calling process to identify this particular socket. As is known in the art, a handle is code that includes access control information and/or a pointer to an object. For example, the socket descriptor appears in "read" and "write" statements and designates the input stream from which the host reads data or the output stream to which the host writes data.

The value of the "format" argument may indicate the address format of a communications domain. For example, as is known in the art, a value of AF\_INET, a designator of the IP 58 address family, may indicate that the socket is to be used to communicate between network devices over an IP 58 connection. Additionally, the value of AF\_INET indicates that the network devices have thirty-two bit addresses (i.e., corresponding to familiar IP 58 address dotted decimal notation w.x.y.z). Similarly, a value of AF\_INET6 may indicate that communication is over IP version 6, which requires one-hundred twenty-eight bit addressing.

The values of the "type" and "protocol" arguments may indicate whether communication over the socket is connectionless or connection oriented. For example, as is known in the art, a value of SOCK\_STREAM for the "type" argument may indicate that communication is a connection oriented virtual circuit by means of TCP 62. Similarly, a value of SOCK\_DGRAM for the "type" argument may indicate that communication is connectionless via datagrams by



means of UDP 60. As is known to those skilled in the art, the "protocol" argument may indicate a particular protocol to control the communication.

Once the kernel of the operating system creates the data structure for the socket, the socket may be bound to the network layer interface and assigned a network address and/or a port number. The "bind" reentrant function typically performs this task for servers and UDP 60 clients. A "bind" function call associates a name with the socket descriptor and is illustrated below:

bind (sd, host address, length) (2)

The socket descriptor, "sd," is the value that was returned by the "socket" call referenced above (1) when the socket data structure was created. The "host address" argument is typically a pointer to a data structure stored in the kernel of the operating system for the host. The data structure typically designates an address family, e.g. AF\_INET, an address for the host, and a port number for the socket. The value of the "length" argument is typically the size of the data structure referenced by the "host address" argument. For example, if the host network device is a server, the "host address" data structure may specify an IP 58 address and a port address provided these are valid addresses for the network address interface 94. Alternatively, the "host address" data structure may specify a well-known port number but not specify an associated IP 58 address. In this case, the kernel of the server would automatically bind the socket to the existing IP 58 address 94 of the network layer interface. Once server processes have bound addresses to sockets, they may announce the socket names to client processes, such as processes 82-86.

Alternatively, if the host network device is configured as a TCP 62 or UDP 60 client, both the values of the host IP 58 address and port number may remain unspecified. In this case, the kernel of the client may automatically bind the socket to the existing IP 58 address of the network address interface 94 and to an ephemeral port of a transport layer interface.

5           Instead of using the “bind” function, however, a TCP 62 or UDP 60 client may also bind a socket using a “connect” reentrant function. The “connect” reentrant function associates the local client’s socket with a target socket on the server. A “connect” call associates a target address with a local socket descriptor as illustrated below:

connect (sd, target address, length) (3)

0051735-02250400   The socket descriptor, “sd,” is the value that was returned when the client’s socket data structure was created by a call to a “socket” function. The “target address” argument is typically a pointer to another data structure stored in the host client. This “target address” data structure typically designates an IP 58 address for the server to which the client is connecting, and a port number for the target socket on the server. The value of the “length” argument is typically the size of the data structure referenced by the “host address” argument. The “connect” function call typically refers to the address structure for the server’s socket, not the client’s socket, and leaves the assignment of the host IP 58 address and port number for the client’s socket to client’s kernel.

20           Once a socket has been created with a “socket” function call and bound with a “bind” or “connect” function call, a process may determine a local network address and port number by a call to a “getsockname” reentrant function. A “getsockname” function call associates an address structure with the socket descriptor and is illustrated below:

getsockname (sd, address, length) (4)

0051735"0225  
004400

The socket descriptor, "sd," is the value that was returned when the socket data structure was created by a call to a "socket" function. This descriptor refers to the socket whose address we wish to determine. The "address" argument is typically a pointer to a data structure stored in the kernel of the operating system for the host. The kernel fills the "address" data structure with the IP 58 address and port number that the socket is currently using. The value of the "length" argument is typically the size of the data structure referenced by the "address" argument. The "getsockname" function is useful whenever other processes 82-86 need to know the actual values for the IP 58 address of the network address interface 94 and the port number of the transport interface. For example, when a server creates and binds a socket it may need to advertise the address of this socket to client network devices. A process on the server may get the socket's address and port number and pass this information on to the client. In this manner, the clients would have access to a target address for connecting to the server's socket.

When network communications no longer need a socket, a process typically de-allocates the socket by a "close" reentrant function. A "close" function call releases a connection for data communications over the network. Previous "connect" or "bind" function calls may have established the connection. The "close" function call is illustrated below:

close (sd) (5)

Both the socket descriptor and port number associated with the socket disappear from the protocol stack 50. The port number of the transport interface typically returns to a pool of port numbers. The network address for the network layer interface 94, however, does not necessarily disappear from the protocol stack 50 because it may be bound to a data-link interface.

## Multiple network address interfaces

In the above description of the socket API, there was only one available network address interface 94 as depicted in FIG. 3. In new and emerging communication systems, such as those accomplishing Network Address Translation ("NAT") or Voice-over-IP ("VoIP"), different processes on the same host may be required to have different IP 58 addresses. NAT is described in IETF RFC 1631 and IETF RFC 2663, the contents of which are incorporated herein by reference. VoIP is described in ITU Recommendation H.323, the contents of which are incorporated herein by reference. As is known in the art, Recommendation H.323 defines negotiation and adaptation layers for video and audio over packet switched networks that do not offer guaranteed service or Quality of Service ("QoS").

FIG. 4 is a block diagram illustrating a stack implementation 100 with multiple network addresses. As is illustrated in FIG. 4, each socket in a multiple network address interface may use a different IP 58 address, as illustrated by IP 58 addresses X, Y, and Z. Additionally, all IP 58 addresses may be bound to the same data-link interface 96. For example, Socket #1 108 is bound to IP 58 address @X 114, Socket #2 110 is bound to IP 58 address @X 116, and Socket #3 112 is bound to IP 58 address @X 118. All the IP 58 interfaces 114-118 are bound to the same data-link interface 96. However, the present invention is not limited to the network and data-link interfaces as illustrated in FIG. 4 and other multiple address interfaces may alternatively be used.

In order to support multiple network addresses, as illustrated in FIG. 4, a host protocol stack 50 may require modification to support multiple IP 58 addresses in a single network layer. Furthermore, the protocol stack 50 may require modification to receive multiple IP 58 addresses

10 dynamically, through a mechanism such as DHCP 66. Alternatively, assignment of the multiple  
network addresses may occur by selecting them from a pool of static IP 58 addresses.  
Additionally, when an application requests a new network address, the protocol stack 50 may  
request the network address from a network address server, receive the network address  
5 assignment from the server, and associate that network address with the socket. It should be  
understood that the present invention is not limited to these configurations for supporting  
multiple network addresses and that many more configurations are possible but the foregoing is  
preferred.

10 In accordance with a preferred embodiment, a new socket API may be constructed that  
directs the kernel of the operating system to establish and use multiple network addresses in the  
protocol stack 50. The new socket API behaves differently compared to the old socket API  
described above; namely, that the new API permits multiple network addresses whereas the old  
API only permits a single network address.

10 Although the corresponding functions in the new API are different in execution and  
produce different results from those of the old API, they may be conceptually analogous to the  
old socket API functions. A programming convention, known to those skilled in the art, is to  
name new socket functions after analogous old socket functions. The naming convention is a  
mnemonic device used by programmers for purposes of replacing old versions of functions with  
new versions. As described below, the new API may contain a modified "socket" system call to  
20 request that a particular socket allocate a new IP 58 address. Additionally, as further described  
below, the new API may contain modified "bind," "connect," "getsockname," and "close"  
functions.

## Modified socket function

A modified "socket" reentrant function that accommodates multiple network addresses may be defined in a variety of ways. In one exemplary preferred embodiment of the modified "socket" function, the modified "socket" function includes an argument to direct the kernel to assign a new network address to a newly created socket. For example, the function call may take the form:

$$sd = \text{socket}(\text{format}, \text{type}, \text{protocol}, \text{tuple}) \quad (6)$$

where the value of the extra "tuple" argument may indicate to the kernel that this socket is to be assigned a new network address from a pool of available addresses (i.e. a static assignment) or dynamically through some mechanism such as DHCP. A default value for "tuple" may indicate that the socket is to be assigned a common network address by the protocol stack in a manner similar to the single address "socket" call referenced above (1).

Another exemplary preferred embodiment uses a new address family for the "format" argument of the socket function call to indicate that the kernel assigns a new network address to the socket. For example, the modified "socket" function call may retain the form of Equation 1. However, a value such as AF\_INET\_TUPLE for the "format" argument may indicate to the kernel that it should assign a new IP address to this socket. It should be understood that the present invention is not limited to these exemplary embodiments. Although not currently known but within the capabilities of one skilled in the art, other ways of creating a modified socket with a new network address are possible.

## Modified bind function

The new socket API may include a modified "bind" reentrant function that accommodates multiple network addresses. A socket that is created by the above-described modified "socket" function call, and has asked for a new network address, will be bound to the new network address by a call to the modified "bind" function. The form of the modified "bind" function may be analogous to the old "bind" function referenced above (2).

The modified "socket" function call has already indicated to the kernel that a new network address is associated with the socket descriptor. The modified "bind" function call passes the socket descriptor, "sd," to the kernel of the operating system for use in the protocol stack 50. The kernel recognizes this value for socket descriptor as associated with a new network address by a previous modified "socket" function call. As before, for the single network address interface, the calling process may specify a well-known port number as a parameter of the bind function call. Alternatively, the process may leave the port number unassigned, in which case the kernel may fill in an ephemeral port number as happened in the old "bind" function. The IP 58 address argument may typically be left unassigned in the "bind" function call.

The kernel of the operating system examines the memory associated with the protocol stack 50 and fills in the new IP 58 address that is associated with the socket descriptor, "sd." In one exemplary preferred embodiment, the kernel selects the IP 58 address from a pool of reserved addresses (i.e. static assignment). Alternatively, the kernel requests the address dynamically from an address server when the "socket" function call creates the socket. The IP 58 address may be stored in a table in memory associated with the protocol stack 50 section of the kernel, along with the socket descriptor. When a process later makes a modified "bind"

function call with the socket descriptor, the kernel searches for this socket descriptor. The kernel determines the previously reserved IP 58 address from the table, and returns this address in the IP 58 address structure of the modified "bind" function.

In another exemplary preferred embodiment, the kernel does not select the IP 58 address until a process calls the modified "bind" function. The call to the modified "socket" function records that the returned socket descriptor, sd, is slated to be assigned a new IP 58 address. The kernel, however, does not yet assign the network address. When the process makes the modified "bind" function call, the kernel determines that the socket descriptor argument contains a socket descriptor that was slated for a new IP 58 address. The kernel selects an IP 58 address for the socket from a pool of addresses or dynamically as described above. The kernel also associates the socket descriptor with the IP 58 address, e.g. in a table, and passes the new IP 58 address up to the IP 58 address structure in the modified "bind" function. It should be understood that the present invention is not limited to these exemplary embodiments and that many other ways of binding a modified socket with a new network address are possible.

#### **Modified connect function**

The "connect" reentrant function also requires modification to accommodate multiple network addresses. As described above, both TCP 62 and UDP 60 clients use a "connect" function call to allocate an IP 58 address and an ephemeral port. The modified "connect" function may retain the form of the old "connect" function call referenced above (3). The target IP 58 address and port number (the address of the server associated with the client) are passed as arguments to the modified "connect" function as before.



The modified "connect" function call passes the socket descriptor, "sd," to the kernel. The kernel recognizes this value for the socket descriptor as being associated with a new network address by a previous modified "socket" function call. As before, for the single network address interface 94, the calling process may allow the kernel to bind an ephemeral port number to the socket.

When a process makes a "connect" function call, the kernel also binds the socket to the new IP 58 address that is associated with the socket descriptor, "sd." In one exemplary preferred embodiment, the kernel selects the IP 58 address from a pool of reserved addresses (i.e. a static assignment). Alternatively, the process may request the address dynamically from an address server (not shown in FIG. 4) when the "socket" function call creates the socket. The IP 58 address may be stored in a table in memory associated with the protocol stack 50 section of the kernel, along with the socket descriptor. A later modified "connect" function call with the socket descriptor causes the kernel to search for this socket descriptor and determine the previously reserved IP 58 address from the table. In this manner, a connection is established between the client socket, having the new IP 58 address and port number, and the server socket, having the target IP 58 address and port number.

In another exemplary preferred embodiment, the kernel does not assign an IP 58 address until the call is made to the modified "connect" function. The call to the modified "socket" function records that the returned socket descriptor, sd, is slated to be assigned a new IP 58 address. The kernel, however, does not yet assign the network address. When a process makes the modified "connect" function call, the kernel determines that the socket descriptor argument contains a socket descriptor that slated for a new IP 58 address. The kernel may select an IP 58

address for the socket from a pool of addresses or, alternatively, the IP 58 address may be assigned dynamically such as is described above. The kernel also associates the socket descriptor with the IP 58 address, e.g. in a table. In this manner, a connection is established between the client socket, having the new IP 58 address and port number, and the server socket, having the target IP 58 address and port number. It should be understood that the present invention is not limited to these exemplary embodiments and that many other ways of connecting a modified socket with a new network address are possible.

### Modified getsockname function

The "getsockname" reentrant function also requires modification to accommodate multiple network addresses. The modified "getsockname" function may retain the form of the old "getsockname" function as referenced above (4). Although the modified "getsockname" function has the same name as the old "getsockname" function, its operation is different. A process calls the modified "getsockname" function and passes the value of the socket descriptor to the kernel. The kernel determines whether the socket descriptor is associated with a new IP 58 address. If the socket descriptor is associated with a new IP 58 address, the kernel fills in the referenced address structure with the new IP 58 address and port number. In this manner, the modified "getsockname" function returns the new network address assigned to the socket in an address data structure. The new network address may have been associated with the socket descriptor during a previous modified "socket" function call, a previous modified "bind" function call, or a previous modified "connect" function call.

### Modified close function

The "close" reentrant function is preferably also modified to accommodate multiple network addresses. The modified "close" function may retain the form of the old "close" function as referenced above (5). A process calls the "close" function and passes the value of the socket descriptor to the kernel. The kernel determines whether this socket descriptor is associated with a new IP 58 address. The new network address may have been associated with the socket descriptor during a previous modified "socket" function call, a previous modified "bind" function call, or a previous modified "connect" function call. If the socket descriptor is associated with the new IP 58 address, the kernel also de-allocates this new IP 58 address and port number. The kernel accomplishes this, for example, by deleting the socket descriptor and/or the new network address from a table in the protocol stack 50 section of the kernel. In this manner, the modified "close" function releases the data communications connection, and the network and virtual interface both disappear. Alternatively, if the kernel assigned the network address to the socket through a dynamic process such as DHCP 66, the socket may automatically close at the end of a lease time by methods known to those skilled in the art.

### Method for using multiple network addresses

FIG. 5 is a flow diagram illustrating a method 130 for using multiple network addresses for interprocess communication through a common physical layer. The method 130 includes creating a first interprocess communication data structure associated with a first network address on a first network device 14 at step 132. The first network device 14 may be the telephony device as illustrated in FIG. 1, although other types of network devices are possible and the present invention is not limited to the telephony device of FIG. 1. An example of an interprocess

communication data structure associated with a network address is a socket that is assigned a new IP 58 address when it is created, as described above. A call to a "socket" function from a process may create the first socket as a data structure in memory associated with the kernel of the first network device 14. For example Socket #1 108 of FIG. 4 may be created by a modified  
5 "socket" function call from Process B 84. However, it should be understood that the present invention is not limited to sockets and IP 58 addresses, and that other types of interprocess communication data structures and network addresses may be used.

At step 134, a first communication is established between the first network device 14 and a second network device 16 using the first interprocess communication data structure and the  
10 first network address. For example, the first socket may establish a connection with a target socket on the second network device 16. This may initiate a data flow between the process bound to the first socket and another process on the second network device 16 bound to the target socket. The communication may be connection oriented, such as a TCP 62 virtual connection, or connectionless, such as a UDP 60 virtual connection. The data flow may follow a modified  
15 "bind" or "connect" function call with the "socket descriptor" for the first socket. Process B 84, for example, may initiate a data flow through the data structure of Socket #1 108. The kernel associated Socket #1 108 with its own network interface 114 having the new IP 58 address @X. However, it should be understood that the present invention is not limited to TCP/IP communications or the like and that other forms of communications are possible, such as UDP/IP  
20 or X.25 communications familiar to those skilled in the art.

The first communication passes through the common physical layer for the first network device 14. As illustrated in FIG. 4, the multiple IP 58 addresses at the network layer share a

common data-link and device driver 96 for the first network device 14 at its physical layer. Data flows to and from all processes 82-86 through the same physical layer interface 96. Data to or from one process 82-86 may go to one or more socket data structures 108-112. The respective network layer interface 114 for the first socket 108 encapsulates or decapsulates the data and processes the packets through the common data-link and physical layer 96.

A second interprocess communication data structure is created at step 136. The second interprocess communication data structure is associated with a second network address on the first network device. The second network address is different from the first network address. For example, Process B 84, having already created Socket #1 108, may also create a second socket, Socket #2 110, by another modified "socket" function call. Socket #2 110 has a different IP 58 address, @Y, compared to Socket #1 108, @X, as the kernel assigns a new IP 58 address when the process makes a modified "socket" function call. There are now two sockets on the first network device 14, and each socket is associated with a different IP 58 address.

At step 138, a second communication is established between the first network device 14 and a third network device (not shown) using the second interprocess communication data structure and the second network address. The second socket may then establish a connection with another target socket on a third computer to provide another virtual connection. This may initiate a data flow between the process and a third process on the third network device bound to the target socket. The communication may be connection oriented, such as a TCP 62 virtual connection, or connectionless, such as a UDP 60 virtual connection. The data flow may follow a modified "bind" or "connect" function call with the "socket descriptor" for the second socket. Process B 84, for example, may initiate a data flow through the data structure of Socket #2 110.

The kernel associated Socket #2 110 with its own network interface 116 having the new IP 58 address @Y. The respective network layer interface 116 for the second socket 110 encapsulates or decapsulates the data and processes the packets through the common data-link and physical layer 96. In this manner, a host computer, application, process, or other entity may support multiple network addresses and bind each address to a separate socket and/or a separate process. This allows a network device to communicate with other network devices using two or more network addresses.

In the present invention, more than one IP 58 interface may map to the same physical or logical data-link device. However, each interface will only send and receive data on behalf of a related group of processes as illustrated in FIG. 4. An advantage of the present invention compared to a traditional data-link interface is that it represents an IP 58 address that is bound to a given executing instance of an application (e.g., a process or related group of processes). Such a configuration may be useful for differentiating IP 58 data traffic to or from a particular host based on something other than a transport-layer parameter such as a TCP 62 or UDP 60 port.

For example, in Internet telephony it may be advantageous for each new call to be ascribed a new IP 58 address. Calls are typically IP 58 messages exchanged between telephony devices in a virtual private network. The telephony devices may typically support multiple calls, each call being associated with a process in the telephony device. In this case, the new IP 58 address resides in a private network address space for the virtual private network. NAT tunnels the calls across the (public) Internet. Ascribing a new and unique IP 58 address to each call may ensure that the identity of the caller is hidden as the call traverses the Internet. Each call process

should be associated with a respective private IP 58 address. The present invention may assign a new IP 58 address to each new call in the manner described above.

Another advantage of supporting multiple IP 58 addresses is improving switching efficiency. As described above, IP 58 addresses rather than TCP 62 or UDP 60 port number may distinguish traffic for applications. In general, layer 3 (network layer) switching using IP 58 addresses is expected to be more efficient than layer 4 (transport layer) switching using TCP 62 or UDP 60 ports. Improved switching efficiency may increase the call capacity of the Internet.

Yet another advantage of using multiple IP 58 addresses is to differentiate data traffic associated with different Quality of Service ("QoS"). QoS provides statistical guarantees of throughput, delay, delay variation, and packet loss. QoS is important in the transmission of Internet telephony, multimedia, and video data streams as these transmissions require a guaranteed bandwidth to function. Different applications on the same host computer may need to communicate with a separate QoS. For example, Internet telephony applications may require a constant or guaranteed bitrate QoS, whereas a web browsing application may require a basic QoS. Differentiating QoS may also improve the ability of the Internet to carry many forms of data communication simultaneously.

Edge routers (24,26) supporting QoS are required to process IP 58 packets differently depending on the QoS of the data communication. Instead of examining the internal details of an incoming IP 58 packet to determine the appropriate QoS, the edge router (24,26) may simply recognize that an IP 58 address in the header is associated with a particular QoS and process that packet accordingly. As discussed above, layer 3 switching is typically expected to be efficient. Ascribing an IP 58 address to each application may allow the IP 58 address for the application to